TRW Systems Engineering &
Development Division

TRW 1

TRW-TS-91-03

Pilot Command Center Testbed Development Environment: A Better Way to Develop C^3 Systems

Charles R. Grauling

September 1991

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

TRW Technology Series

chnology Series

Pilot Command Center Testbed Development Environment: A Better Way to Develop C^3 Systems

Charles R. Grauling

TRW Systems Integration Group
Redondo Beach, California

Abstract

TRW has recently completed a research project sponsored by the USAF's Electronics Systems Division (ESD) to support the development of the Pilot Command Center (PCC) for SDI. The PCC is a facility that will be used by the USAF to develop the system level requirements for the Communication, Command and Control (C^3) elements of SDI with particular emphasis on determining the role of the human in control by means of experimentation. TRW's role in this project was to evaluate the feasibility of reusing the software architecture developed on an earlier ESD sponsored command center development project for this application. The PCC Testbed Development Environment (PTDE) which is the project's primary product provides the capability to rapidly implement and integrate prototype C^3 system application software (including message sets, databases, mission algorithms, and interactive displays) and to easily migrate the prototype software into a complete C^3 system testbed suitable for conducting realtime experiments. The experience gained in developing and using the generic command center tools and techniques that were used can be applied to future command center development programs to reduce system development costs and schedules. This paper summarizes the PTDE design and development process with supporting rationale. It also includes some "lessons learned" and recommendations for enhancing the development process on future C^3 development efforts.

Background

The foundation technology for the PTDE's generic command center approach was developed on ESD's Command Center Processing and Display System Replacement (CCPDS-R) program starting in 1986. The CCPDS-R program includes the replacement of all the ADPE and C^3 software in a set of related, existing command centers. The requirement to simultaneously develop software for multiple command centers created extra incentives to incorporate reusable components and development techniques in the design process. The emergence of Ada as a viable programming language with its support of concurrency and software engineering enabled us to implement reliable, reusable Network Architecture Services (NAS) software [Royce 1989] as the infrastructure for all the command centers supported by CCPDS-R. This software is currently being reused on other TRW command center development projects. Given NAS as a foundation, the CCPDS-R program developed tools that it used to automatically generate a substantial portion of its newly developed Ada source code. These tools and techniques were built specifically for the CCPDS-R program and are not designed to be generally reusable. However, the basic ideas behind them can be used as the basis for developing an integrated generic command center capability which promises to greatly enhance expected productivity on future command center development projects [Grauling 1990].

The effectiveness of the NAS framework and the use of automated code generation have been major contributors to the success enjoyed by the CCPDS-R program. These principles and techniques can be applied on other command center development activities by using them as a basis for designing generic C^3 system application components that execute within the NAS provided framework. The CCPDS-R implementation was optimized for performance in CCPDS-R's target environment (DEC VAX/VMS, 1987 vintage display technology, etc.). Broadening the applicability of CCPDS-R

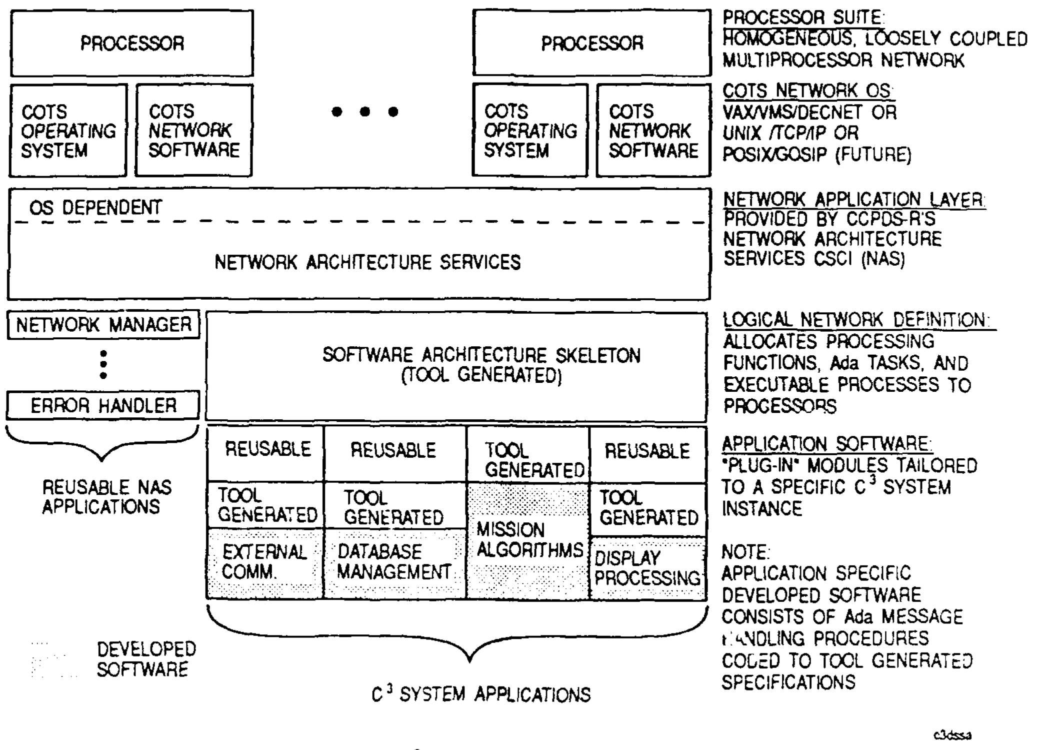

PROCESSOR
PROCESSOR
COTS OPERATING SYSTEM
COTS NETWORK SOFTWARE
COTS OPERATING SYSTEM
COTS NETWORK SOFTWARE
PROCESSOR SUITE:
HOMOGENEOUS, LOOSELY COUPLED MULTIPROCESSOR NETWORK
COTS NETWORK OS:
VAX/VMS/DECNET OR UNIX /TCP/IP OR POSIX/GOSIP (FUTURE)
OS DEPENDENT
NETWORK ARCHITECTURE SERVICES
NETWORK APPLICATION LAYER:
PROVIDED BY CCPDS-R'S NETWORK ARCHITECTURE SERVICES CSCI (NAS)
NETWORK MANAGER
ERROR HANDLER
REUSABLE NAS APPLICATIONS
SOFTWARE ARCHITECTURE SKELETON (TOOL GENERATED)
LOGICAL NETWORK DEFINITION:
ALLOCATES PROCESSING FUNCTIONS, Ada TASKS, AND EXECUTABLE PROCESSES TO PROCESSORS
REUSABLE
REUSABLE
TOOL GENERATED
REUSABLE
TOOL GENERATED
TOOL GENERATED
MISSION ALGORITHMS
TOOL GENERATED
EXTERNAL COMM.
DATABASE MANAGEMENT
DISPLAY PROCESSING
APPLICATION SOFTWARE:
"PLUG-IN" MODULES TAILORED TO A SPECIFIC C³ SYSTEM INSTANCE
NOTE:
APPLICATION SPECIFIC DEVELOPED SOFTWARE CONSISTS OF Ada MESSAGE HANDLING PROCEDURES CODED TO TOOL GENERATED SPECIFICATIONS
DEVELOPED SOFTWARE
C³ SYSTEM APPLICATIONS
c3dssa

Figure 1: C^3 System Software Architecture

technology requires modifying the implementation to optimize for portability and adherence to industry standards. This would allow the benefits of the CCPDS-R architecture while providing the capability to take advantage of emerging commercial off the shelf technology such as processors, workstations and standard software interfaces. The development of the PTDE provided an excellent opportunity to identify and implement these modifications. The remainder of this section is a brief synopsis of NAS and CCPDS-R's software development approach as background for readers who are unfamiliar with them.

NAS Fundamentals. NAS is an outgrowth of an Internal Research and Development (IR&D) project to determine the applicability of Ada to C^3 system development which was started in 1985. The IR&D's first output was a product called InterTask Communication (ITC). ITC provides a mechanism which enables Ada tasks to perform logical I/O amongst themselves without explicit synchronization or knowledge of the underlying hardware. Once ITC was in place, it became apparent that we could layer additional reusable executive software components on top of ITC to create an integrated development and runtime environment for implementing distributed applications (Figure 1). ITC and the executive layer above it were collected into a product called Network Architecture Services (NAS) to provide a complete executive framework for building complex applications on a distributed computer system. The functions performed or supported by NAS include: system initialization, fault detection, reconfiguration, error handling, and interactive network health and status monitoring. NAS is currently in use successfully on CCPDS-R [Royce 1989].

ITC is the foundation capability upon which NAS has been built. The two critical communication abstractions defined by ITC are called sockets and circuits. A socket is a virtual I/O channel that an application task uses to send and/or receive messages. Application tasks can create and delete sockets at runtime using ITC provided services. Application tasks communicate by establishing connections among their sockets called circuits. Once a task has created its sockets and circuits, it is able to perform logical I/O with other tasks by reading messages from its input sockets and writing messages to its output sockets. ITC provides the message buffering and synchronization necessary to allow both the sending and receiving tasks to execute without waiting. A system consists of independently executing

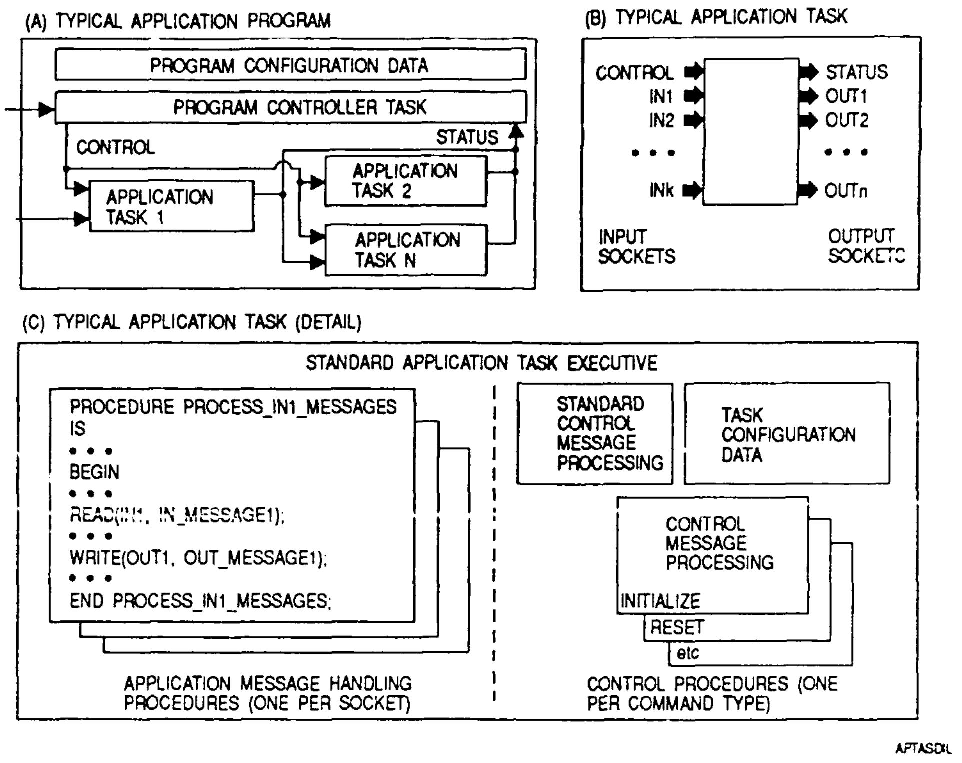

(A) TYPICAL APPLICATION PROGRAM
PROGRAM CONFIGURATION DATA
PROGRAM CONTROLLER TASK
CONTROL
STATUS
APPLICATION TASK 1
APPLICATION TASK 2
APPLICATION TASK N
(B) TYPICAL APPLICATION TASK
CONTROL
IN1
IN2
INk
STATUS
OUT1
OUT2
OUTn
INPUT SOCKETS
OUTPUT SOCKETS
(C) TYPICAL APPLICATION TASK (DETAIL)
STANDARD APPLICATION TASK EXECUTIVE
PROCEDURE PROCESS_IN1_MESSAGES
IS
BEGIN
READ(IN1, IN_MESSAGE1);
WRITE(OUT1, OUT_MESSAGE1);
END PROCESS_IN1_MESSAGES;
STANDARD CONTROL MESSAGE PROCESSING
TASK CONFIGURATION DATA
CONTROL MESSAGE PROCESSING
INITIALIZE
RESET
etc
APPLICATION MESSAGE HANDLING PROCEDURES (ONE PER SOCKET)
CONTROL PROCEDURES (ONE PER COMMAND TYPE)
APTASDIL

Figure 2: Standard Application Program Structure

tasks whose only stimuli are messages which they read and whose only outputs are messages which they write.

Designing a system under ITC requires partitioning the application into software modules that conform to this model. We call this design methodology "message based design". Message based design provides complete encapsulation of the multiprocessor architecture details within ITC (i.e., application tasks send and receive messages of predefined types without any knowledge of the location of the receivers or senders). The advantages of this approach include:

- It enforces hardware independent design by requiring the exclusive use of logical input/output. Hardware independence contributes to the overall flexibility of the software design.
- It promotes a cohesive and modular software design by requiring early formal definition of task-to-task interfaces and disallowing common coupling among tasks.
- It reduces Ada training costs and development risk by encapsulating Ada tasking within ITC. Designers can implement multitasking applications with all the benefits of using Ada (e.g., packages, strong typing) without explicitly using Ada's relatively complex tasking mechanism.
- It reduces synchronization problems among communicating application tasks. Sending tasks never have to synchronize with receiving tasks to accomplish communication. They simply send. ITC buffers the messages and delivers them to the receiving tasks upon request. The synchronization required is accomplished using Ada tasking within ITC.

Although ITC provides a sufficient capability for implementing message based systems, it is possible to standardize the structure of ITC based applications software. The standard task executive handles certain complex system issues such as system fault detection, error handling, and system reconfiguration in a manner that is as transparent to applications as possible.

Figure 2 (A) is a schematic representation of a typical NAS based standard application program. NAS standardizes the implementation of executive functions by including NAS provided control processing in every standard application program. This processing behaves as if it were performed by an included program controller task as illustrated. The program control task performs processing that is common to all NAS application programs such as process synchronization upon initial program load, ITC network login, standard error handling, status reporting, and monitoring the health and status of the application tasks contained in the program.

The standard application task, Figure 2 (B), is the basic processing module for a NAS based system. It consists of the standard executive shell, standard control message processing, and application specific message handling and control procedures shown in Figure 2 (C). The executive in each application task detects when ITC has a message to deliver to one of its input sockets and determines which socket has the next message to read. For each message arriving on an application input socket, the task shell executes the application message handling procedure associated with that socket. The application message handling procedure then reads and processes the message. This processing generally includes producing output such as writing one or more messages to one or more of its output sockets or performing external I/O. This is how the task accomplishes useful work.

Incoming control messages are handled by NAS. Certain control messages are handled in a fashion that is transparent to the application task. The control message that requests a standard task level status information is an example of a transparent control message. These messages provide the mechanism whereby the program controller can gather performance data or detect failures (i.e., a response time-out). NAS's control message mechanism is also extendable to allow application specific control procedures. This feature is used to allow applications to issue network wide commands (for example to command a network reconfiguration or a database reset) without requiring explicit knowledge of the network's structure. In addition to providing the top level abstraction for the implementation of application components, the existence of standardized application components provided a framework to include support functions such as standardized error handling, performance monitoring and logging, interactive network control, and system reconfiguration in all NAS based systems. The standard structure of NAS based systems makes it possible to automate portions of the software development process. One of the first tools in this category was the Software Architecture Skeleton (SAS) Builder tool. This tool allows one to build a text file description of an entire NAS network. The tool processes the text file and produces all the source code required to build the network out of the generic NAS application components.

CCPDS-R Software Development Approach. In addition to using NAS, the CCPDS-R project used a new approach to the software development process described in [Springman 1989] and [Royce 1990]. This approach uses incremental builds, top down integration, and tools which automatically generate a substantial fraction of the deliverable Ada source code. The tools were developed specifically for the CCPDS-R project in an ad hoc fashion (i.e., they were invented and implemented by programmers for their own purposes to aid configuration management and source code generation wherever it made sense). Our approach to demonstrating the feasibility of reusing CCPDS-R technology in the PCC included using NAS and its associated SAS builder tool "as is", reusing CCPDS-R application components wherever appropriate, replacing custom CCPDS-R components with equivalent Commercial Off the Shelf (COTS) technology where possible, and enhancing the ad hoc code generating tools to make them more effective for use in the PTDE.

PTDE Design Overview

The PTDE consists of the two major components illustrated in Figure 3. The C^3 system development environment is a host facility with ADPE resources and software development tools that can be used to build working prototypes of C^3 systems. The C^3 system evaluation environment provides the ADPE resources and application software necessary to conduct command center experiments with human controllers performing realistic command center tasks in a realistically simulated command center environment. The key attribute of the C^3 system development environment is its ability to support all aspects of rapid command center development. This includes facilities for rapidly building display prototypes, developing the software required to handle external message sets and the various types of format conversion that they require, simulating external C^3 system components (e.g., external weapon and sensor systems), and developing the general purpose software required to implement mission specific algorithm processing. The existing CCPDS-R development system was the starting point for the development of this portion of the PTDE. The key attribute for the C^3 system evaluation environment is its realism. Realism comes from providing a simulated command center environment that has adequate interactive response time performance and uses simulation techniques with sufficient fidelity to properly exercise

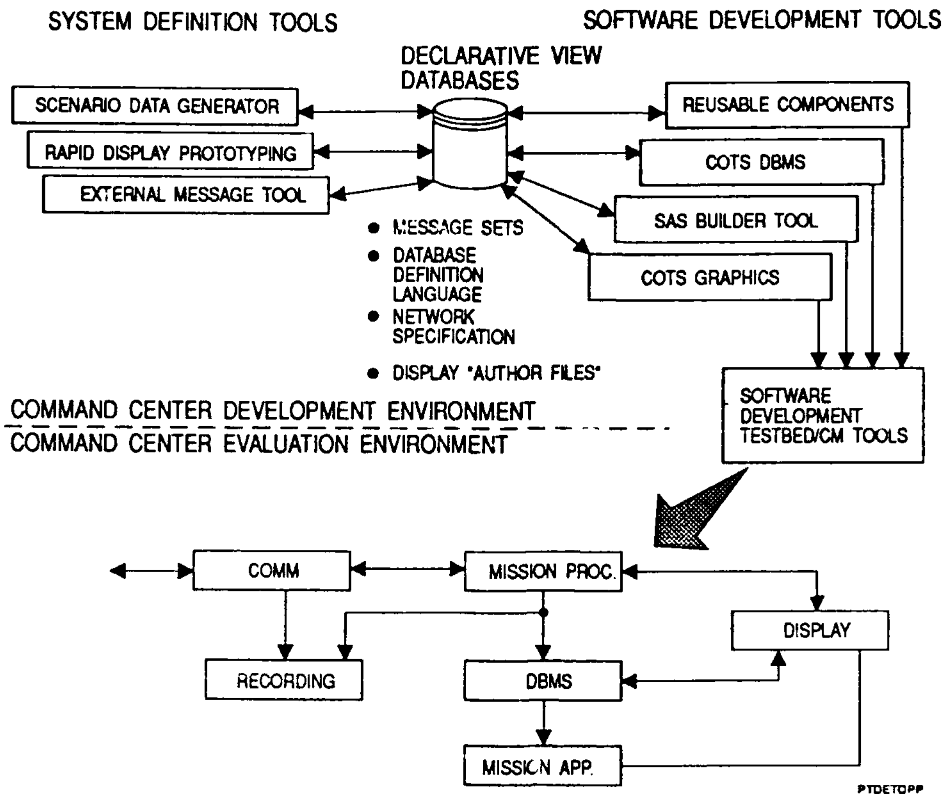
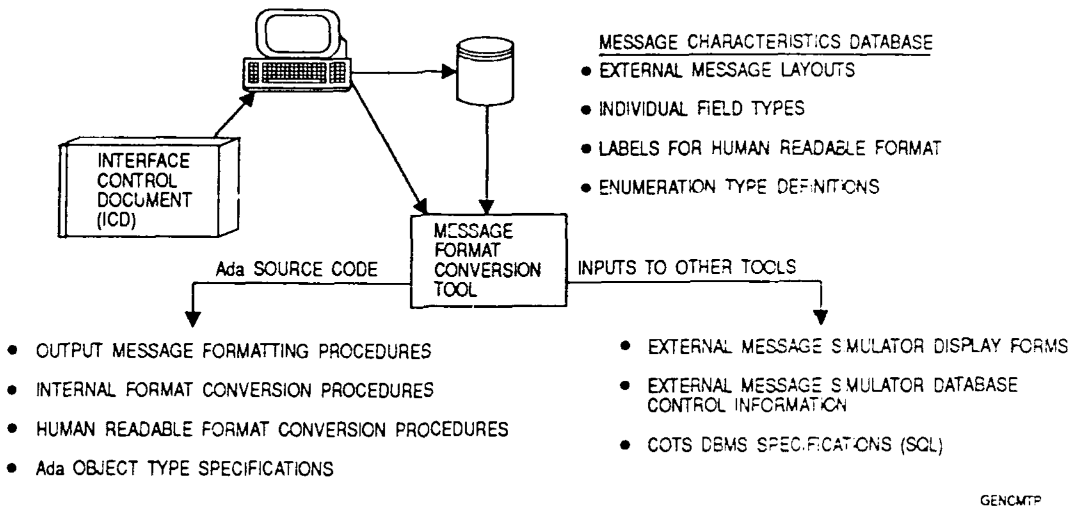


Figure 3: PTDE Top Level Architecture

the humans in control. Another important feature of the C^3 system evaluation environment is its ability to gather the measurable performance information required to quantitatively assess the operation of the experimental command centers. The CCPDS-R runtime environment with NAS and its associated performance measurement facilities were the starting point for the development of this capability.

The PTDE development process consisted primarily of the integration of existing C^3 software components with very little new software development. Its design is based on the software engineering principles and supporting reusable components that were developed on the CCPDS-R program and other IR&D projects, and the use of Commercial Off the Shelf (COTS) products. We used an incremental development approach consisting of three builds. The first build started with a baseline system taken directly from the CCPDS-R development facility. We modified it to remove certain capabilities that are not necessary in the PCC environment. These capabilities include the primary/shadow threads that CCPDS-R uses to provide hardware fault tolerance and the Test/Real separation that the CCPDS-R program uses to allow test mode operation on the real system. The implementation of these capabilities is transparent to most application software in a NAS based command center because of NAS's built-in support for these functions. The performance impact of having these capabilities are negligible in any realistic command center application because they use redundant processing resources. The second build consisted of replacing the display and database management systems that were developed specifically for CCPDS-R with COTS software products (e.g., a commercially available relational DBMS and a commercially available X-windows implementation). The motivation for this approach was to trade performance for flexibility and portability. Since the PTDE is an experimental facility whose mission is develop C^3 system requirements, it is more important to be able to rapidly implement systems than to squeeze the most performance out ADPE hardware. The third build was the formal demonstration of the PTDE's ability to accommodate the transition from the CCPDS-R application to Space Defense Systems (SDS) applica-



MESSAGE CHARACTERISTICS DATABASE
• EXTERNAL MESSAGE LAYOUTS
• INDIVIDUAL FIELD TYPES
• LABELS FOR HUMAN READABLE FORMAT
• ENUMERATION TYPE DEFINITIONS
INTERFACE CONTROL DOCUMENT (ICD)
MESSAGE FORMAT CONVERSION TOOL
Ada SOURCE CODE
INPUTS TO OTHER TOOLS
• OUTPUT MESSAGE FORMATTING PROCEDURES
• INTERNAL FORMAT CONVERSION PROCEDURES
• HUMAN READABLE FORMAT CONVERSION PROCEDURES
• Ada OBJECT TYPE SPECIFICATIONS
• EXTERNAL MESSAGE SIMULATOR DISPLAY FORMS
• EXTERNAL MESSAGE SIMULATOR DATABASE CONTROL INFORMATION
• COTS DBMS SPECIFICATIONS (SQL)
GENCMTP

Figure 4: External Message Tool

tions. The primary objective of the third build was to go through the process of installing an SDS specific algorithm involving weapons and sensors. In the process, we measured the productivity characteristics of the PTDE's C^3 system development environment. The remainder of this section provides descriptions of the major components of the PTDE with emphasis on the tradeoffs that had to be made among various software quality attributes such as portability, performance characteristics, usability and flexibility. It is organized in terms of the basic functions that are common to all C^3 systems: external message interfaces, database management, mission algorithm processing, and user interfaces.

External Representation Management. One of the fundamental capabilities of a C^3 system is its ability to receive incoming message traffic, convert them from their external format to a variety of internal formats so they may be used by processing and display application software, and to generate outbound messages in an agreed upon external format. Real world command centers generally deal with hundreds of different types of messages and a small number of different external communication protocols. The high leverage area is clearly in the message formatting portion of the communication processing. The CCPDS-R program developed an automated system for producing format conversion procedures. This system involves capturing the external format definition information usually provided in an external Interface Control Document (ICD) in a text file and using a tool to parse this text file and automatically generate key software products. The text file serves as the source file for automatically generating the Ada source code and the other internal representations that are required in a typical C^3 system shown in Figure 4. The list of products produced by the Communication Message Tool (CMT) for a typical input message type consists of the following items:

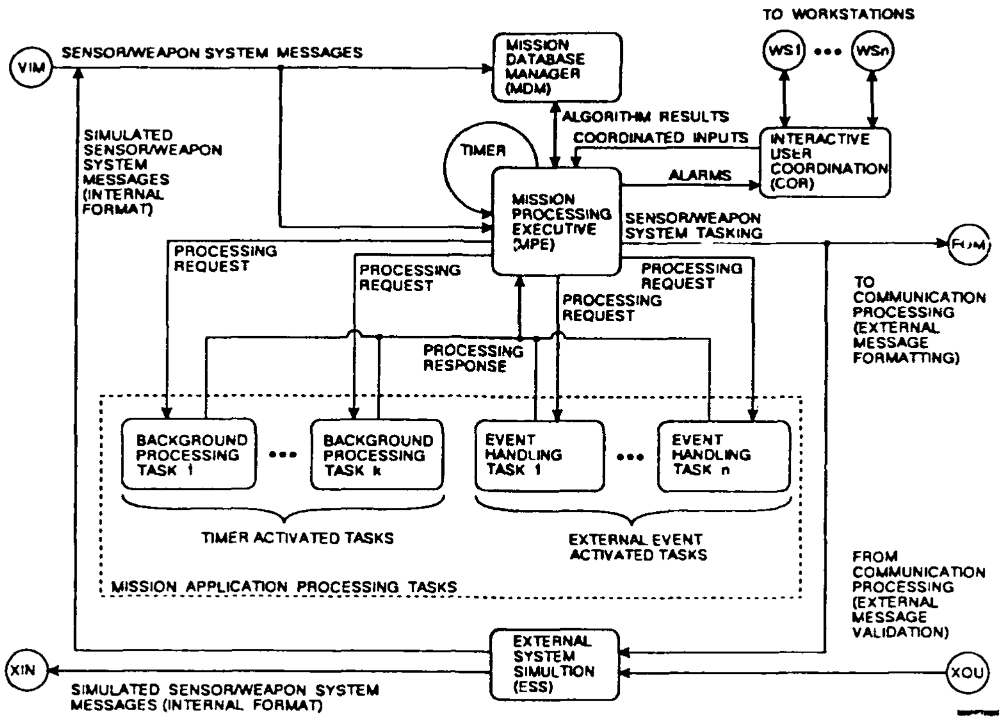
- Message Validation Procedures. Every external message that enters the system must be picked apart on a field by field basis. Each field must be checked for violations of range of value constraints that are specified in the ICD prior to conversion into its internal format. The message tool produces the source code for an Ada procedure to perform this processing for each input message type. In some cases, special ad hoc processing may be required as specified in the ICD. The message tool also provides the hooks to add this type of application specific field validation logic to the format conversion procedure if required.
- Ada Representation Specifications. In the CCPDS-R software architecture, external messages enter the system through communication processing tasks that perform protocol processing and produce an input datagram for each arriving external message. The bit by bit description of this format is defined in the ICD. The Ada programs that process these messages need an Ada representation clause to locate individual fields within the input message datagram. One of the first steps in the processing of a message is its conversion from an array of bytes to an Ada ob-

ject with an associated internal Ada record type. The type specification for the internal Ada object representation of each input message type is automatically produced by the message tool based on information extracted from the ICD and incorporated in the message tool's input text file.

- Human Readable Output Formatting Procedures. CCPDS-R allows the user to print any external message in human readable form on a suitable output device (such as a line printer). The CCPDS-R message tool can automatically generate the Ada source code for a procedure to convert the internal representation of an external message into its human readable form. It does so based on the information in the CMT source text file which contains field definitions and other related information such as the agreed upon field labels and mnemonics.
- External Message Simulator Display Forms. CCPDS-R has a built in external message simulation capability that allows the operator to construct and subsequently inject externally formatted test messages. This capability uses an on-line database of form descriptions to support the operator interface to the external message generation capability. The CMT automatically populates this database using information contained in its source text file.
- External Message Simulator Control Information. This item is also related to the external message simulation capability. The internal simulation capability uses an on-line database to control last minute message formatting details such as inserting the correct time tags in simulated external messages. The CMT automatically populates this database using information contained in its source text file.
- COTS DBMS Schema Definition. C^3 systems routinely log incoming and outgoing messages. If a COTS DBMS is used to implement this function, the ability to easily perform on-line data review and reduction is provided by the COTS product. The information necessary to automatically generate the SQL data definition statements which define a table for each type of external message already exists in the CMT's source text file. Adding this type of support required a minor modification to the existing CCPDS-R CMT.

The key advantage provided by CMT is its consistency. When the external definition of a given message type changes, a single update of the input source text and subsequent rerun of the tool will produce a complete and consistent set of modified source code for all affected software components and databases. The absence of this type of tool in the PTDE would mean that PTDE users would have to manually create all the above mentioned source code for each new message type that is added to the PTDE's repertoire of supported messages. The CCPDS-R CMT is adequate for CCPDS-R's purposes, but it needed enhancement in the form of a more user friendly front end to make it more suitable for use in the PTDE. In its original state, CMT did no error checking of its input source file. Keypunch errors or failure to adhere to the rigid flat file format could cause the CMT to crash (i.e., raise an unhandled exception at run time). This front end helps the user create CMT input by providing a form filling type of interface which also performs on-line data integrity checking. It makes the process of creating new message types and updating old message types more productive for the occasional user.

Database Management. In our generic command center model, external messages arrive, populate a database and cause mission algorithms to run. Mission algorithms generate processed information which is also stored in the database. They can also cause operator alarms which will generally cause the C^3 system operators to take some action. Operators interact with the database through their workstations under the control of user interaction software. The database management function provides the storage, distribution, and integrity protection of the displayable database. There are several different implementation strategies for this important function. The choice of a design approach involves the tradeoffs among attributes such as development cost, performance, and flexibility. The CCPDS-R program was performance driven with a well known, stable list of required displayable data items. It therefore selected a high efficiency memory resident database approach over a COTS DBMS based approach. In CCPDS-R, there was a single master database which was the recipient of all messages and algorithm results. The CCPDS-R master database actively distributed displayable database updates to each of the workstations which were maintaining a local memory resident copy of the master database. Automatic updates to users' displays were triggered by database distribution events. This approach ensured display consistency across the complete set of workstations that were serviced by the system. These properties were explicitly specified in the CCPDS-R system specification.



TO WORKSTATIONS
WS1 ••• WSn
VIM
SENSOR/WEAPON SYSTEM MESSAGES
MISSION DATABASE MANAGER (MDM)
ALGORITHM RESULTS
COORDINATED INPUTS
INTERACTIVE USER COORDINATION (COR)
SIMULATED SENSOR/WEAPON SYSTEM MESSAGES (INTERNAL FORMAT)
TIMER
ALARMS
MISSION PROCESSING EXECUTIVE (MPE)
SENSOR/WEAPON SYSTEM TASKING
FOM
PROCESSING REQUEST
PROCESSING REQUEST
PROCESSING REQUEST
PROCESSING REQUEST
TO COMMUNICATION PROCESSING (EXTERNAL MESSAGE FORMATTING)
PROCESSING RESPONSE
BACKGROUND PROCESSING TASK 1
BACKGROUND PROCESSING TASK k
EVENT HANDLING TASK 1
EVENT HANDLING TASK n
TIMER ACTIVATED TASKS
EXTERNAL EVENT ACTIVATED TASKS
MISSION APPLICATION PROCESSING TASKS
FROM COMMUNICATION PROCESSING (EXTERNAL MESSAGE VALIDATION)
EXTERNAL SYSTEM SIMULTION (ESS)
XIN
XOU
SIMULATED SENSOR/WEAPON SYSTEM MESSAGES (INTERNAL FORMAT)

Figure 5: Typical Mission Processing SAS Structure

The requirements are quite different for the PTDE. In particular, flexibility and ease of change are more important than performance, provided the interactive response time is realistic for the purposes of determining operator workload. We therefore decided to replace the CCPDS-R database management system with a functionally equivalent COTS DBMS based implementation. This meant replacing the master and slave database managers in the CCPDS-R implementation with a general purpose database server. Under this approach, application processes that create displayable data store their data in tables managed by the server while application processes requiring access to data throughout the network can log into the server as clients. The database server handles all the processing required to ensure data integrity. The major advantages of a COTS based approach are the additional facilities that it provides. These include forms editors, interactive user query support, report writers, security features, and support for data archiving and recovery. Under the original CCPDS-R approach, all access to the display database was performed via procedure calls. The PTDE's system has the same interface with access to the underlying database system completely encapsulated in the bodies of the access procedures. Support for data definition is built into the CMT. The amount of source coding required to implement the access procedures in the body parts is roughly the same in both approaches. In the CCPDS-R system, this coding is done in Ada whereas the COTS DBMS based approach uses vendor provided bindings to access procedures or SQL module language.

Mission Processing. In the generic command center model, the mission processing functional area contains the algorithms that process incoming information and send displayable results to the database management system. These results are ultimately used by the interactive users as decision aids. Since mission processing is application unique by definition, it is the functional area of a command center that is least likely to benefit from the use of generic command center design techniques. We applied some general guidelines for packaging mission processing into NAS compatible networks to create the mission processing portion of the Version 3 software architecture skeleton, Figure 5.

The main objective in the design of the Version 3 mission processing software architecture skeleton is to provide a flexible structure into which mission processing algorithms could easily be inserted. In particular, the ability to spread computationally intense processing over multiple processors in order to ensure that currently unspecified performance requirements could be satisfied by simply adding processors without major software modifications. Figure 5 illustrates the general structure of the mission processing portion of the Version 3 SAS. This structure features a top level controller task which handles all input events (such as an incoming sensor message or an operator entered control command) and controls the execution of a collection of event handler tasks by routing messages to the appropriate event handler task as a function of the incoming message type and the current state of the system. The basic idea behind this task structure is to provide a task framework that allows the software architect to allocate processing associated with specific weapon and sensor systems to separate tasks. These separate tasks could then be executed on different processors.

In this architecture, the Mission Processing Executive task functions primarily as a top level problem solver. It implements top level algorithms (i.e., algorithms whose logic depends on the results generated by multiple weapon or sensor system models). It also has the capability to initiate and control concurrent processing functions. This capability enables it to function as an ADPE processing load leveler. For example, it is possible to replicate a given type of event handler task on multiple processors to allow algorithms with high CPU resource demands to be broken up and spread over multiple processors under the Mission Processing Executive's control. Under the message based design paradigm, all algorithm processing is triggered by incoming message traffic. There are three types of mission processing trigger messages: external communication message receipt, internal timer events, and interactive user service requests. In general, any of these events will cause the Mission Processing Executive (MPE) task to perform some form of algorithm processing. Throughout this paper, the term "algorithm processing" refers to the processing performed in response to any of these external events.

Algorithm processing normally results in an update to the dynamic database which is used to populate user displays and/or generate outbound messages. MPE performs part of this processing directly (i.e., within its own input message handling procedures) and it may also perform part of it concurrently by sending control messages to one or more of the event handler illustrated in Figure 5. The relationship between MCE and the other application tasks that it controls is similar to the client/server model commonly used in COTS DBMS and workstation products, tailored as appropriate to operate in the concurrent programming model provided by NAS. In the PTDE Version 3 SAS architecture, the client task (MCE) sends an ITC message to one or more of the event handler tasks which in turn behave as servers. These tasks subsequently perform the requested processing, optionally update displayable database items, and return a response message to MCE upon completion. The response message contains status information and optional algorithm processing results.

In the Ballistic Missile Defense (BMD) application, the server tasks are typically models of individual sensor or weapon systems, or in some cases, specific platforms within a sensor or weapon system because simulation is a fundamental component of the battle management process. For example, evaluating the BMD system's probability of negating a missile attack involves using some sort of model to simulate ballistic missiles, sensor systems, and weapons systems. Likewise, evaluating candidate engagement opportunities invariably involves modeling specific ballistic missile trajectories and weapon platform physical characteristics (such as orbits and end game capabilities). The general guidelines that we used to create this software architecture skeleton are suitable for application in any command center application. The notion of a creating a top level control task and partitioning the number crunching functions into separable NAS application tasks is generally applicable to any command center application. It is also useful to be able to allocate the external simulation processing functions to separate tasks in order to easily be able to move them into external processors. This will make it possible to more accurately observe the performance characteristics of the main command center processing equipment by putting the external simulation processing in separate processors.

User Interface. The basic principles of providing software developers with higher level abstraction supported by text file driven tools that generate the Ada source code were also used in the User Interface functional area on the CCPDS-R project. The primary user interface on CCPDS-R was a workstation constructed out of a DEC Microvax processor which drove a graphics processor. The choice of this hardware was effectively dictated by the end user's interactive graphics responsiveness requirements. The hardware was chosen in 1987, before the emergence of the today's high speed workstations. One of the goals of the PTDE development was to replace the CCPDS-R workstations with a functionally equivalent capability based on X-windows. The primary reason for this objective was to achieve

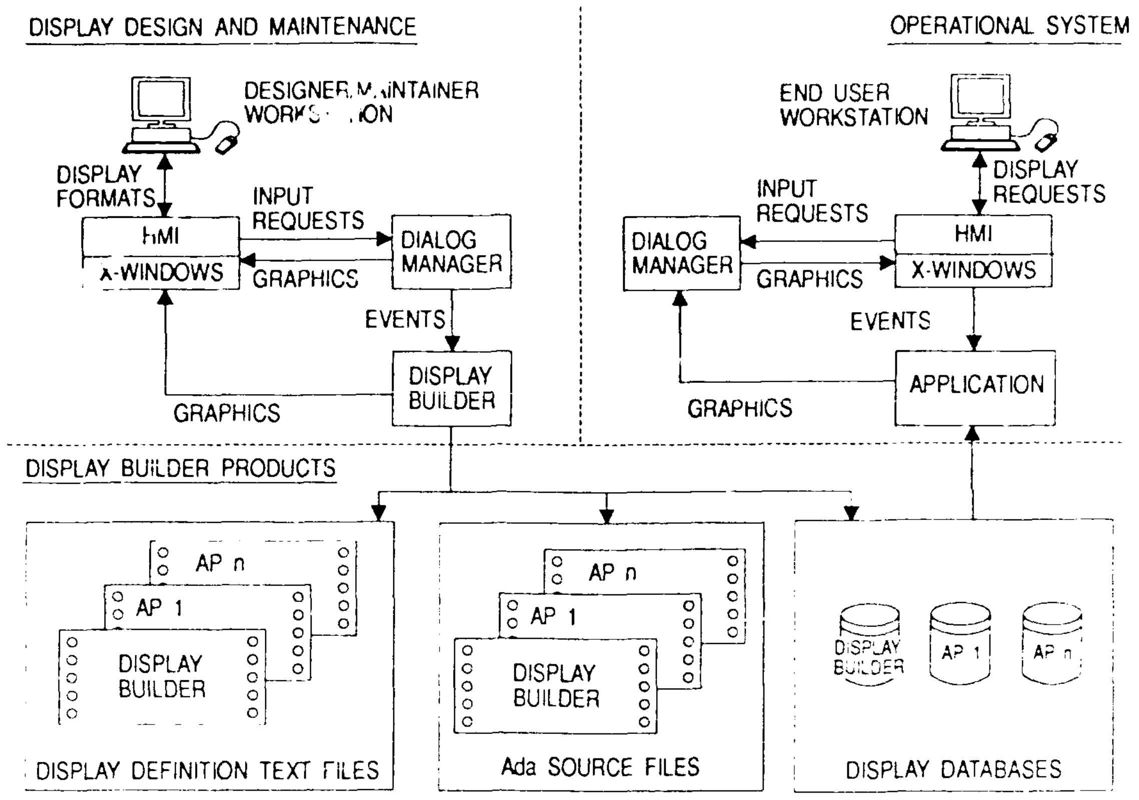


Figure 6: Implementing User Interfaces With Display Builder

the device independence to take advantage of future inexpensive, high performance workstation technology.

The design of the user interface for C^3 systems is traditionally a long and difficult activity involving interaction and concurrence among representatives of the end user, system engineering, human factors, and software development communities. It invariably includes iteration on the layout of user interface screens and prototypes of their operation when integrated with application processing. It was therefore necessary to include a rapid user interface prototyping facility as part of the PTDE The PTDE's display builder, illustrated in Figure 6 uses software that had been developed on earlier user interface IR&D projects. It supports user interface design iteration by providing a way to rapidly build both display prototypes (graphical representations of the way user interaction screens will look) and interactive application prototypes (executable Ada source code that implements a complete set of interactive displays including underlying user interaction). The best way to explain this point is to step through the PTDE's user interface design process with an emphasis on the role of the PTDE's display builder tool.

The first step in designing the user interface is the human engineering step. This involves analyzing the overall mission requirements, the command center's operational concept, and the applicable human factors standards that must be incorporated the user interface. The output of this step is the definition of the requirements on the human interface implementation. These requirements include the definition of the complete set of user interaction screens, the information content of each screen, and the interaction mechanisms (such as menus, forms, pickable items, etc.) that must be available on each screen. The PTDE does not provide any specific tools to support this activity. It is primarily an analytical process as opposed to a software development activity. However, the output of this step is the input to the display prototype development process.

The user interface implementor uses the PTDE's display builder to interactively create an applications complete user interface (such as the hierarchy of windows including all the background maps, tabular display layouts, pull-down menus, etc.). The display builder automatically generates the needed internal representations such as Ada source code and the internal data structures that are used by the workstation software to perform display generation when the application runs in the operational environment. The display implementor works with a set of high level abstract objects called "widgets". The PTDE's widget set is based on standard Motif widgets including pull-down menus, pushbuttons, toggle buttons, and radio buttons. It is a general purpose widget set which also includes some missile defense specific windows for use as map backgrounds and annotation symbols. The display implementor needs only to learn the semantics of these widgets because the PTDE's tools hide much of the underlying graphics implementation system (in this case X-windows and Motif toolkits). The PTDE's display builder is actually just another application built on top of a general purpose interactive application framework. In this case the application is to prototype interactive user interfaces as opposed to performing interactive command and control. The display builder generates Ada source code and resources database necessary to compile and run the operational version of each application it creates. It also generates a text file description of the application which can be used to port an application to some other hardware environment.

The display prototypes are generally used to review and attain concurrence on display layout details. Modifying displays based on user comments can usually be accomplished in a matter of minutes using the PTDE's display prototype building capabilities. User interactions can be checked out using a display builder test mode which allows one to step through the menu tree and exercise an application's various buttons and forms. Application specific processing is not available at this point in the process. Once concurrence on the look and feel of the an application's user interface is achieved, the next major step in the process is to build the application prototype. The application prototype is based on a standard Ada process main program that executes in the workstation during on-line command center operation (indicated as the "Application" in Figure 6). This program contains the main application event handler augmented to handle additional external events including the arrival of ITC messages from a network of NAS application programs. The PTDE's user interface implementation facility generates the source code for a skeleton of this program for an entire application which includes stubs for the application specific call back events. This approach is analogous to the way in which the SAS builder tools automatically generates Ada source code for the complete set of processes for a NAS network. Of course, the application developer must replace the stubs with appropriate application specific processing for call back events such as menu picks and form entries. This is analogous to the ITC message handling procedures that are the plugged into a NAS based software architecture skeleton. The key point here is that the PTDE provides the capability to automatically generate all the display system related source code (including application call back stubs) for an entire interactive application that interfaces with a NAS network. The system includes an Ada graphics library which allow the application developer easily perform any additional application specific graphics programming that may be required.

Conclusion and Recommendations

The Version 3 demonstration included a simple BMD application that included 23 new external messages, 13 new internal database objects, 3 new mission processing algorithm procedures, and a user interface consisting of 29 new displays. It contained over 100,000 new lines of Ada source code. The demonstration was developed and integrated over a four month period by a team of five people who also produced user level and interface description documentation. The software development productivity experienced on this project is superior to other projects performing comparable development without the benefit of comparable tools.

The PTDE development project was completed by the end of March, 1991. The development team is currently continuing to develop and use these tools and techniques on a command center development project using PTDE technology for the USAF's Space Command in Colorado Springs. Recently implemented enhancements to the PTDE based system include: migration to the next generation of NAS (called UNAS [Royce 1991a,b]) for improved vendor independence and portability, performance enhancements in the database management area, and new display builder capabilities. The display builder is currently being used by Air Force personnel to design the user interfaces on the new project.

The PTDE development effort has provided a valuable opportunity to work with new technology that can substantially improve the productivity of the command center development process. The most important lesson learned is the value and feasibility of using automation to enhance Ada software development productivity. The PTDE development team started with homemade tools that were inherited from the CCPDS-R program and enhanced them by adding better user interfaces. The tools generally handle the mundane chores necessary to manage the declarative part of a complex system. For example, the SAS builder tool deals with the static declaration of a task network's topology, the CMT deals with the static declaration of external message set attributes, and the display builder deals the user interface attributes. Tools of this nature are not hard to build. The original tools were developed by CCPDS-R application developers for their own use on one project. They were not called out as contract deliverables. Their spartan nature is a result of the fact that there is no room for anything else in a cost constrained development program. The PTDE development gave us a valuable opportunity to enhance the existing tools for more general usage. The task is by no means complete. Using homemade tools naturally generates more ideas for enhancements.

The homemade approach to tool building that we used is not the only way to apply these ideas today. There are many currently available commercial products that one could use accomplish the same ends. We looked at [illegible] alternatives for implementing the display builder and settled with the homemade approach for its flexibility, performance, and ability to support producing all Ada applications. Organizations not having an equivalent legacy would be likely to find a COTS based approach preferable for their needs.

Acknowledgements

This work could not have been done without the support of USAF ESD and the foundation created by the men and women of the CCPDS-R project. All the real work on the PCC project has been done by a dedicated team in Colorado Springs lead by Jim Franklin and consisting of Rhonda Davis, Ya Shu Feng, Phil Gage, Jeff Gerhart, Dave Kayser, J. R. Johnson, and Bill Watts.

Biography

Charles Grauling is a Software Chief Engineer for TRW's Systems Engineering and Development Division. He received his BS in Electrical Engineering from Cornell University in 1966, MS in Electrical Engineering from the Massachusetts Institute of Technology in 1968, and MS in Computer Science from the University of Southern California in 1972. He has been responsible for software requirements analysis and architecture design on CCPDS-R and other C^3 projects at TRW since 1982. He is currently working on the application the ideas expressed in this paper to the problem of reengineering existing Management Information Systems to take advantage of today's distributed processing and workstation technology.

References

[Grauling 1990] Grauling, C. R., "Network Architecture Services: An Environment for Constructing Command, Control and Communication Systems", *Second IEEE Workshop on Future Trends of Distributed Computing Systems Proceedings, Cairo, Egypt, October 1990.*

[Royce 1989] Royce, W. E., "Reliable, Reusable Ada Components For Constructing Large, Distributed Multi-task Networks: Network Architecture Services (NAS)", *TRI-Ada '89 Proceedings, Pittsburgh, October 1989.*

[Royce 1991a] Royce, W. E., "TRW's Ada Process Model for Incremental Development of Large Software Systems", *Proceedings of the 12th International Conference on Software Engineering, Nice, France, March 26-30, 1990.*

[Royce 1991b] Royce, W. E. and Brown, D. L. "Architecting Distributed Realtime Ada Applications: The Software Architect's Lifecycle Environment", *Ada IX Proceedings, March 1991*

[Royce 1991] Royce, W. E. et al. "Universal Network Architecture Services: A Portability Case Study", *Ada IX Proceedings, March 1991*

[Springman 1989] Springman, M. C., "Incremental Test Approach for DOD-STD-2167A Ada Projects," *TRI-Ada '89 Proceedings, Pittsburgh, October 1989.*